

NAG Numerical Functions for GPUs Function Document

nag_gpu_sobol_normal

1 Purpose

nag_gpu_sobol_normal generates multidimensional quasi-random sequences from a Normal distribution with mean 0.0 and variance 1.0.

The initialization function `nag_gpu_sobol_init` must be called prior to the first call to **nag_gpu_sobol_normal**. Successive calls to **nag_gpu_sobol_normal** will generate successive values along the sequence. Once all desired values have been obtained, the function `nag_gpu_sobol_cleanup` must be called to free allocated system resources.

The dimension of the Sobol sequence is given by the `dim` parameter in `nag_gpu_sobol_init`. All references to `dim` below refer to that value.

2 Specification

```
#include <nag_gpu.h>
```

```
void nag_gpu_sobol_normal (int npts, float *d_P, Nag_Gpu_Comm *comm)
```

```
void nag_gpu_sobol_normal (int npts, double *d_P, Nag_Gpu_Comm *comm)
```

3 References

None.

4 Arguments

1: **npts** – int *Input*

On entry: the number of quasi-random values required.

Constraint: $1 \leq \text{npts}$.

2: **d_p[npts × dim]** – float * *Output*

3: **d_p[npts × dim]** – double * *Output*

This overloaded parameter can take type double or float.

The array must be allocated in the *device memory space*.

On exit: the `npts` quasi-random values of dimension `dim` from a Normal distribution. For the n th Sobol point $x_n = (x_n^1, x_n^2, \dots, x_n^{\text{dim}})$ with $0 \leq n < \text{npts}$, the d th dimension x_n^d will be stored at location `d_P[(d-1)*npts+n]`. In other words, the first `npts` values correspond to dimension 1, the second `npts` to dimension 2, and so on.

4: **comm** – Nag_Gpu_Comm * *Communication Data*

`Nag_Gpu_Comm` is a NAG defined type which holds state and communication information and must not be modified in any way.

The structure initialized by a previous call to `nag_gpu_sobol_init`.

5 Error Indicators and Warnings

No argument constraint checking is carried out by this function. You can insert a call to function `nag_gpu_utilCheckMsg` (contained in the header file `nag_gpu.h`) following the call to **nag_gpu_sobol_normal** to check for the last CUDA error message, if any, generated during execution.

6 Example

This example prints the first 64 quasi-random numbers of dimension 5 from a Normal distribution generated by **nag_gpu_sobol_normal**. The initialization function `nag_gpu_sobol_init` is called first to prepare the Sobol generator.

6.1 Program Text

```

/* nag_gpu_sobol_test
 *
 *
 * Copyright 2010, Numerical Algorithms Group Ltd, Oxford, UK.
 *
 * Version 0.3, 2010.
 *
 */

////////////////////////////////////
// pre-processor includes
////////////////////////////////////
#include <stdio.h>

////////////////////////////////////
// include libraries
////////////////////////////////////
#include <nag_gpu.h>

////////////////////////////////////
// define precision
////////////////////////////////////
#define PRS float

////////////////////////////////////
// Main program
////////////////////////////////////

int main(int argc, char **argv)
{
    FILE *fpout = stdout;
    //fopen("nag_gpu_sobol_normal_example.results.txt","w");

    // Max dims supported by Sobol generator is 50,000
    const int dim = 40;
    // Generate batchSize numbers at a time on the GPU. For large sample
    // sizes this can avoid out-of-memory errors
    const int batchSize = 50;

    // Total number of batches to generate
    const int nBatches = 10;

    PRS
    // Host (PC) storage for batch-generated GPU Sobol numbers
    *h_GpuBatchNums = 0,
    // Device (GPU) storage for batch-generated Sobol numbers
    *d_GpuBatchNums = 0;

    Nag_Gpu_Comm
    // Communication structure pointer for GPU generator
    comm;

    // Scrambling vector for GPU Sobol
    unsigned int *v = 0;
    // Offset into Sobol sequence
    const int offset = 0;

```

```

// Number of dimensions to print: must be less than dim
const int printDim = 5;
// Number of Sobol values to print
int nPrint = 64;
// Make sure we have enough numbers
nPrint =
    (nPrint > batchSize*nBatches
     ? batchSize*nBatches : nPrint);
// Storage for numbers we'll print
PRS *h_print = 0;
// Offset into h_print at which to store GPU numbers
int printOffset = 0;

// Greeting
fprintf(fpout, "\n\n NAG GPU Sobol Example Program: normal\n\n");

// Set device with highest Gflops/s
cudaSetDevice( nag_gpu_GetMaxGflopsDeviceId() );

// Allocate memory
if (!(h_GpuBatchNums = (PRS*)malloc(sizeof(PRS)*dim*batchSize)) ) {
    fprintf(stderr, "h_GpuBatchNums allocation failure\n");
    goto END;
}
if (!(cudaSuccess == cudaMalloc((void **)&d_GpuBatchNums,
    sizeof(PRS)*dim*batchSize))) {
    fprintf(stderr, "d_GpuBatchNums allocation failure\n");
    goto END;
}
if (!(v = (unsigned int*)malloc(sizeof(unsigned int)*dim)) ) {
    fprintf(stderr, "v allocation failure\n");
    goto END;
}
if (!(h_print = (PRS*)malloc(sizeof(PRS)*printDim*nPrint)) ) {
    fprintf(stderr, "h_print allocation failure\n");
    goto END;
}

// Set scrambling vector to 0 - don't want scrambling
for (int i=0; i < dim; i++) {
    v[i] = 0;
}

// Initialise GPU Sobol generator
nag_gpu_sobol_init(dim, offset, v, &comm);

for(int batchCount = 0; batchCount < nBatches; batchCount++) {
    // Launch GPU computation asynchronously
    nag_gpu_sobol_normal(batchSize, d_GpuBatchNums, &comm);

    /*
     * One can now launch other kernels to operate on the Sobol numbers,
     * or copy the numbers to the host and operate on them there.
     * Here we simply copy them to the host in order to print them
     */

    cudaMemcpy(h_GpuBatchNums, d_GpuBatchNums,
        sizeof(PRS)*dim*batchSize, cudaMemcpyDeviceToHost);

    // Copy GPU values to print storage
    for (int i = 0; i < batchSize; i++) {
        if (i+printOffset < nPrint) {
            for (int j = 0; j < printDim; j++) {
                // We transpose the GPU data for easier display
                h_print[(printOffset+i)*printDim + j] =
                    h_GpuBatchNums[i+j*batchSize];
            }
        }
    }
}

```

```

    }
  } // END print storage loop
  printOffset += batchSize;

} // END batching for loop

// Print Sobol numbers
fprintf(fpout,
        "\nThe first 64 GPU numbers from dimensions 1 to 5:\n");
fprintf(fpout, "      n ");
for(int d = 1; d<=printDim; d++) fprintf(fpout, "      dim%-2d ", d);
for(int i = 0; i < nPrint; i++) {
  fprintf(fpout, "\n%5d", i+1);
  for(int d = 0; d < printDim; d++)
    fprintf(fpout, " % 10.4f ", h_print[i*printDim+d]);
}
fprintf(fpout, "\n\n");

END:

// Release GPU and CPU memory
nag_gpu_sobol_cleanup(&comm);

if (d_GpuBatchNums)
  nag_gpu_utilSafeCall( cudaFree(d_GpuBatchNums) );

if (h_GpuBatchNums)
  free(h_GpuBatchNums);

if (v)
  free(v);

if (h_print)
  free(h_print);

fflush(stdout);
fflush(stderr);

if (fpout!=stdout)
  fclose(fpout);

cudaThreadExit();
return 0;
}

```

6.2 Program Data

None.

6.3 Program Results

NAG GPU Sobol Example Program: normal

The first 64 GPU numbers from dimensions 1 to 5:

n	dim1	dim2	dim3	dim4	dim5
1	-5.4200	-5.4200	-5.4200	-5.4200	-5.4200
2	0.0000	0.0000	0.0000	0.0000	0.0000
3	0.6745	-0.6745	-0.6745	-0.6745	0.6745
4	-0.6745	0.6745	0.6745	0.6745	-0.6745
5	-0.3186	-0.3186	0.3186	1.1503	-0.3186
6	1.1503	1.1503	-1.1503	-0.3186	1.1503
7	0.3186	-1.1503	1.1503	0.3186	0.3186
8	-1.1503	0.3186	-0.3186	-1.1503	-1.1503
9	-0.8871	-0.4888	1.5341	-0.1573	0.1573

10	0.4888	0.8871	-0.1573	1.5341	-1.5341
11	1.5341	-1.5341	0.4888	-0.8871	-0.4888
12	-0.1573	0.1573	-0.8871	0.4888	0.8871
13	-0.4888	-0.8871	-0.4888	0.1573	1.5341
14	0.8871	0.4888	0.8871	-1.5341	-0.1573
15	0.1573	-0.1573	-1.5341	0.8871	-0.8871
16	-1.5341	1.5341	0.1573	-0.4888	0.4888
17	-1.3180	-0.0784	-0.0784	0.4023	-0.5791
18	0.2372	1.8627	1.8627	-1.0100	0.7764
19	1.0100	-0.7764	-0.7764	1.3180	0.0784
20	-0.4023	0.5791	0.5791	-0.2372	-1.8627
21	-0.0784	-1.3180	1.0100	-0.5791	-1.0100
22	1.8627	0.2372	-0.4023	0.7764	0.4023
23	0.5791	-0.4023	0.2372	-1.8627	1.3180
24	-0.7764	1.0100	-1.3180	0.0784	-0.2372
25	-1.0100	-1.0100	0.0784	1.0100	1.0100
26	0.4023	0.4023	-1.8627	-0.4023	-0.4023
27	1.3180	-0.2372	0.7764	0.2372	-1.3180
28	-0.2372	1.3180	-0.5791	-1.3180	0.2372
29	-0.5791	-0.5791	-1.0100	-0.7764	0.5791
30	0.7764	0.7764	0.4023	0.5791	-0.7764
31	0.0784	-1.8627	-0.2372	-0.0784	-0.0784
32	-1.8627	0.0784	1.3180	1.8627	1.8627
33	-1.6759	-0.6261	0.5334	0.1178	-1.0775
34	0.1178	0.7245	-0.8305	-1.6759	0.3601
35	0.8305	-2.1539	1.6759	0.8305	1.2299
36	-0.5334	0.0392	-0.1178	-0.5334	-0.2777
37	-0.1971	-1.0775	-1.4178	-0.1971	-0.6261
38	1.4178	0.3601	0.1971	1.4178	0.7245
39	0.4451	-0.2777	-0.4451	-0.9468	0.0392
40	-0.9468	1.2299	0.9468	0.4451	-2.1539
41	-0.7245	-1.4178	-0.6261	2.1539	0.5334
42	0.6261	0.1971	0.7245	-0.0392	-0.8305
43	2.1539	-0.4451	-2.1539	0.6261	-0.1178
44	-0.0392	0.9468	0.0392	-0.7245	1.6759
45	-0.3601	-0.1178	1.2299	1.2299	0.9468
46	1.0775	1.6759	-0.2777	0.2777	-0.4451
47	0.2777	-0.8305	0.3601	-0.3601	-1.4178
48	-1.2299	0.5334	-1.0775	1.0775	0.1971
49	-1.4178	-0.7245	0.8305	-1.0775	-0.1971
50	0.1971	0.6261	-0.5334	0.3601	1.4178
51	0.9468	-0.0392	0.1178	-0.2777	0.4451
52	-0.4451	2.1539	-1.6759	1.2299	-0.9468
53	-0.1178	-0.3601	-0.1971	0.7245	-1.6759
54	1.6759	1.0775	1.4178	-0.6261	0.1178
55	0.5334	-1.2299	-0.9468	0.0392	0.8305
56	-0.8305	0.2777	0.4451	-2.1539	-0.5334
57	-1.0775	-0.1971	-0.7245	-0.4451	2.1539
58	0.3601	1.4178	0.6261	0.9468	-0.0392
59	1.2299	-0.9468	-0.0392	-1.4178	-0.7245
60	-0.2777	0.4451	2.1539	0.1971	0.6261
61	-0.6261	-1.6759	0.2777	0.5334	0.2777
62	0.7245	0.1178	-1.2299	-0.8305	-1.2299
63	0.0392	-0.5334	1.0775	1.6759	-0.3601
64	-2.1539	0.8305	-0.3601	-0.1178	1.0775